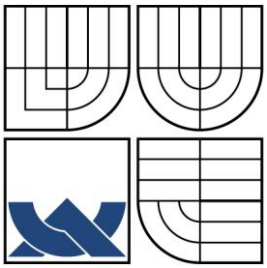
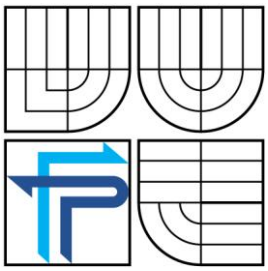
View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Digital library of Brno University of Technology



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA PODNIKATELSKÁ
ÚSTAV INFORMATIKY

FACULTY OF BUSINESS AND MANAGEMENT
INSTITUTE OF INFORMATICS

GRAFY, GRAFOVÉ ALGORITMY A JEJICH VYUŽITÍ PŘI HLEDÁNÍ NEJKRATŠÍ CESTY

GRAPHS, GRAPH ALGORITHMS AND THEIR USE IN FINDING THE SHORTEST PATH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ OTT
AUTHOR

VEDOUCÍ PRÁCE — Mgr. MARTINA BOBALOVÁ, Ph.D.
SUPERVISOR

BRNO 2010

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Lukáš Ott

Manažerská informatika (6209R021)

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách, Studijním a zkušebním řádem VUT v Brně a Směrnicí děkana pro realizaci bakalářských a magisterských studijních programů zadává bakalářskou práci s názvem:

Grafy, grafové algoritmy a jejich využití při hledání nejkratší cesty

v anglickém jazyce:

Graphs, Graph Algorithms and their Use in Finding the Shortest Path

Pokyny pro vypracování:

Úvod
Vymezení problému a cíle práce
Teoretická východiska práce
Analýza problému a současné situace
Vlastní návrhy řešení, přínos návrhů řešení
Závěr
Seznam použité literatury
Přílohy

Podle § 60 zákona č. 121/2000 Sb. (autorský zákon) v platném znění, je tato práce "Školním dílem". Využití této práce se řídí právním režimem autorského zákona. Citace povoluje Fakulta podnikatelská Vysokého učení technického v Brně. Podmínkou externího využití této práce je uzavření "Licenční smlouvy" dle autorského zákona.

Seznam odborné literatury:

DEMEL, Jiří. Grafy a jejich aplikace. 1. vyd. Praha : Academia, 2002. 258 s. ISBN 80-200-0990-6.
MATOUŠEK, Jiří, NEŠETŘIL, Jaroslav. Kapitoly z diskrétní matematiky. 3. upr. vyd. Praha : Karolinum, 2007. 424 s. ISBN 978-80-246-1411-3.
MEZNÍK, Ivan. Diskrétní matematika pro užitou informatiku. 1. vyd. Brno : Cerm, 2009. 104 s. ISBN 978-80-214-3948-1.
ROSEN, Kenneth H. Discrete Mathematics and Its Applications. 4th edition. New York : Mc Graw-Hill, 1999. 832 s. ISBN 0-07-289905-0.

Vedoucí bakalářské práce: Mgr. Martina Bobalová, Ph.D.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2009/10.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

L.S.

Ing. Jiří Kříž, Ph.D.
Ředitel ústavu

doc. RNDr. Anna Putnová, Ph.D., MBA
Děkanka

V Brně, dne 7. 2. 2010

Anotace

Bakalářská práce je zaměřena na seznámení se s teorií grafů a grafových algoritmů pro hledání nejkratší cesty a následnou implementací získaných poznatků do programu MS Excel 2003 s využitím jazyka VBA.

Teorie grafů nás provází vším, od elementárních problémů, až po složité úkony a pokud budeme schopni pochopit základní poznatky uvedené v této práci, budeme schopni je následně využít také v praxi.

Annotation

This Bachelor thesis concentrates on introducing graphs and graph algorithms theories for finding the shortest path and consequential implementation of acquired pieces of knowledge into program MS Excel 2003 using VBA language.

Chart theory applies to everything, from small problems to complex actions. If we are able to understand the basic pieces of knowledge presented in this thesis, we will also be able to put them into practice.

Klíčová slova

Graf, grafový algoritmus, cesta, minimální kostra grafu, VBA, praktické využití.

Key words

Graph, graph algorithm, path, minimum spanning tree, VBA, practical use.

Bibliografická citace

OTT, L. *Grafy, grafové algoritmy a jejich využití při hledání nejkratší cesty*. Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2010. 47 s. Vedoucí bakalářské práce Mgr. Martina Bobalová, Ph.D.

Čestné prohlášení

Prohlašuji, že předložená bakalářská práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a právech souvisejících s právem autorským).

V Brně, dne 27. května 2010

Lukáš Ott

…….

Podpis

Poděkování

Tímto bych rád poděkoval paní Mgr. Martině Bobalové Ph. D. za cenné rady a připomínky, které vedly k vytvoření této bakalářské práce.

Obsah

Úvod a cíle práce

V současné době se svět okolo nás pohybuje velkou rychlostí a je jen otázkou lidského myšlení a využití nabízených technologií, k tomu abychom si co nejvíce zjednodušili život. Neboť kdy než dnes je nejlepší doba k tomu, zamyslet se nad tím, jak rychle se vyvíjí svět kolem nás. Vezměme si například dobu těsně po revoluci, kdy se začínaly u nás objevovat nové firmy a především nové počítačové technologie, které jsou nedílnou součástí pokroku dnešní doby.

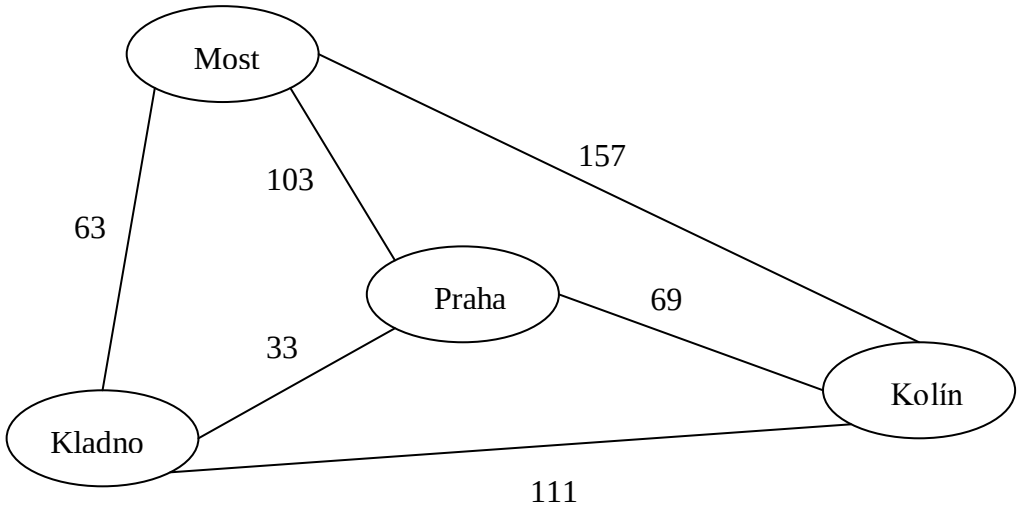
Od doby kdy došlo k vytvoření prvního dopravního prostředku, člověk řešil ve svém podvědomí, i když si to v dřívějších dobách možná až tolik neuvědomoval, jak se dostat jednoduše z bodu A do bodu B v co nejkratší době a pokud možno s co nejnižším vynaložením energie. V době kdy se využívaly pouze papírové mapy, bylo nutné si před cestou zasednout ke stolu a naplánovat si cestu, což ovšem nemuselo znamenat vždy nalezení nejkratší cesty a zároveň příprava zabrala hodně času v závislosti na délce trasy. Proto se v dnešní době využívají GPS technologie jako součást tradičního života. Je ovšem nutné si uvědomit, že celá problematika hledaní nejkratší cesty je spojená především s diskrétní matematikou, která zahrnuje jako jednu ze svých kapitol grafy. Ony již zmíněné grafy se postupem času stále vyvíjely a z mého pohledu se dále také vyvíjet budou.

Práce je rozdělena do dvou částí, do části teoretické, kde se budeme věnovat obecnému seznámení s grafy a s jednotlivými algoritmy na hledání nejkratší cesty a do části praktické, kde popíšeme postup sestavení algoritmu pro hledání minimální kostry grafu v programu MS Excel 2003 za pomocí VBA a jeho následnou implementaci pro praktické použití..

Teoretická východiska práce a současná situace

1 Pojem graf a historie graf

Co je to vlastně graf? Je to jeden z nejdůležitějších pojmů v diskrétní matematice. Graf je tvořen z vrcholů (uzlů), které v našem případě můžeme označit jako města, a hran, představující cesty a vzdálenosti mezi dvěma městy.



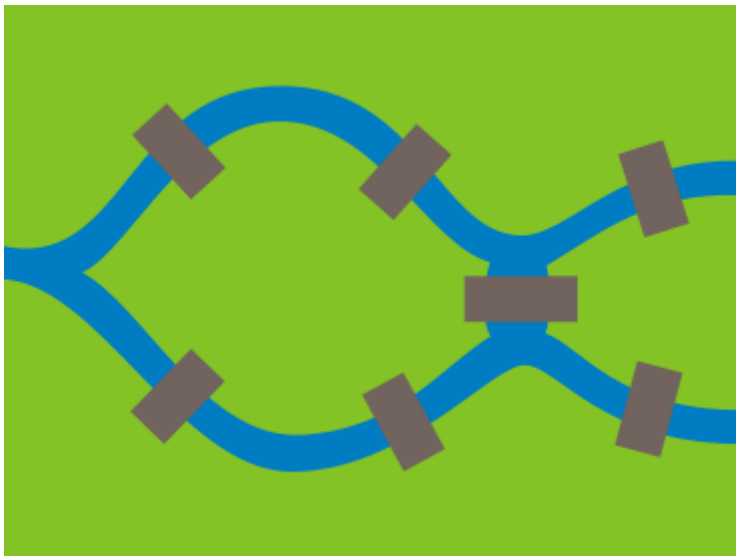
Obrázek č.1: Ohodnocený graf

První zmínka o grafech je spojena se jménem L.Eulera (1707 – 1782), který roku 1736 publikoval řešení příkladu *Sedmi mostů města Königsbergu* (zmíněno v podkapitole 1.1). Další vývoj grafů je spojen především s implementací do různých oborů. Například díky grafům dochází k řešení problémů týkajících se elektrických sítí (Kirkhoff) a chemických isomerů (Cayley). Nejznámější úlohou teorie grafů v 19. století byl problém čtyř barev, kde se řešilo, zda je možné každou mapu obarvit pomocí čtyř barev tak, aby sousední státy měli rozdílné barvy, což se skutečně potvrdilo. Tato úloha byla vyřešena v roce 1976. Ve 20. století bylo dosaženo významných poznatků také na půdě Československa, kdy v roce 1926 publikoval Otakar Borůvka (1899-1995) svůj graf pro nalezení minimální kostry grafu, což vedlo k nejvýhodnější výstavbě

elektrických sítí. Stejný problém ovšem jiným způsobem vyřešil v roce 1930 také Vojtěch Jarník (1897-1970).

1.1 Eulerovský tah

V teorii grafů se Eulerovský tah označuje takový tah, který obsahuje každou hranu grafu právě jednou. Zavedl jej již výše zmíněný Leonhard Euler v roce 1736, když se snažil vyřešit slavný problém *Sedmi mostů města Königsbergu*.



Obrázek č.2: Ohodnocený graf

Tento problém nám lépe pomůže seznámit se s teorií grafů. Cílem tohoto problému byla snaha o to přejít přes každý most právě jednou a vrátit se zpět do výchozího bodu. Euler přišel na to, že není možné po matematické stránce tento problém vyřešit. V současné době to znamená, že zmíněný problém nemůžeme nazývat eulerovským grafem, neboť jej v podstatě není možné nakreslit „jedním tahem".[1)]

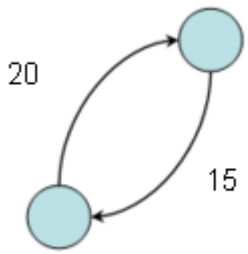
[1] JIROVSKÝ, L. Teorie grafů [online]. 2008 [cit. 2009-11-12]. Dostupný z WWW: <http://teorie-grafu.elfineer.cz/>.

2 Základní druhy grafů a popis hran

Abychom se mohli lépe seznámit se základními druhy grafů, osvětlíme si nejdříve, jaké existují typy hran mezi jednotlivými vrcholy.

- *Orientovaná hrana* – uspořádaná dvojice vrcholů, hranou lze procházet pouze ve vyznačeném směru.
- *Neorientovaná hrana* – neuspořádaná dvojice vrcholů, hranou lze procházet v obou směrech.
- *Násobné hrany* – více hran spojujících stejné vrcholy.
- *Smyčka* – hrana vedoucí z vrcholu do něj samotného.
- *Ohodnocená hrana* – vyjadřuje kvalitu nebo kvantitu vztahu mezi dvěma vrcholy (např. vzdálenost, cenu apod.).

Rovněž je nutné uvědomit si fakt, že uvedené hrany můžeme do sebe také kombinovat různým způsobem. Například můžeme vidět násobnou hranu, která bude zároveň orientovaná a také ohodnocená, záleží na daném příkladu.



Obrázek č.3: Násobná hrana, orientovaná a ohodnocená

Nyní již můžeme přistoupit k popsání základních typů grafů.

2.1 Jednoduchý graf

Pod pojmem jednoduchý graf si nejlépe můžeme představit například telefonické spojení mezi jednotlivými městy, kdy linka funguje obousměrně a zároveň neexistuje linka, která by vstupovala sama do sebe. V tomto případě graf využívá neorientované hrany a dvě města spojuje vždy právě jedna hrana.

2.2 Multigraf

Nastane-li nám situace, že bude potřeba propojit města více než jednou linkou, potom použijeme multigraf, který se skládá z neorientovaných hran a uzlů, a dva různé uzly, může propojovat s více hranami. Každý jednoduchý graf je zároveň multigrafem.

2.3 Pseudograf

Jestliže budeme chtít propojit linku samu se sebou, což se používá například v callcentrech při pohovoru s novými uchazeči o zaměstnání, použijeme pro zobrazení této situace pseudograf. Základní rozdíl mezi třemi prozatím uvedenými typy grafů je ten, že jednoduchý graf neobsahuje násobné hrany ani smyčky, multigraf obsahuje násobné hrany, nikoli však smyčky a pseudograf obsahuje jak smyčky tak také násobné hrany.

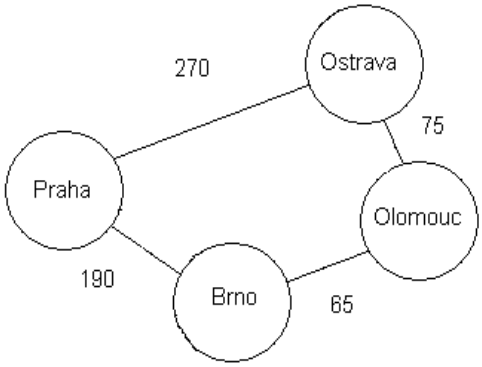
2.4 Orientovaný graf

Pokud budeme dále chtít, aby linka procházela pouze jedním směrem mezi danými dvěma uzly, použijeme pro znázornění orientovaný graf. V tomto případě se připouštějí smyčky, kde je možná pouze jedna orientace. Násobné hrany stejného směru

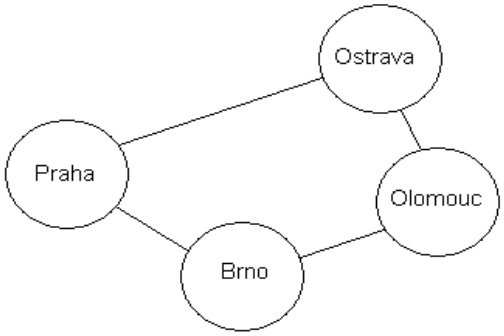
se v tomto případě nepřipouštějí. Pakliže se násobné orientované hrany připouští, jedná se o takzvaný orientovaný multigraf (případně pseudograf).

2.5 Ohodnocený graf

V některých situacích například při řešení problémů spojených se vzdáleností mezi dvěma městy, se používá ohodnocený graf. Může se jednat o grafy orientované, nebo neorientované. Čísla mohou vyjadřovat nejen informace o vzdálenosti, ale také času, průchodu apod.[2)]



Obrázek č.4: Ohodnocený graf



Obrázek č.5: Jednoduchý graf

[2] MEZNÍK, I. Diskrétní matematika pro užitou informatiku. 2009. s.34-37.

3 Hledání nejkratší cesty a pojem algoritmus

Pod pojmem algoritmus si můžeme představit recept. Obsahem receptu je to, co chceme uvařit, přísady, které budeme potřebovat, a posloupnost kroků k dosažení daného výsledku. Měl by být přesně stanoven vstup, tzn. co máme k dispozici, výstup, tzn. čeho chceme dosáhnout, a v neposlední řadě také co máme dělat se vstupem, abychom dosáhli daného výsledku.

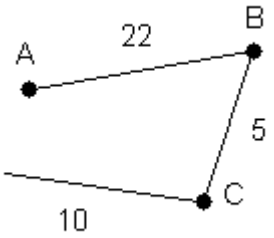
Pokud budeme mluvit o teorii grafů, mezi jednu ze základních algoritmických úloh, bezpochyby patří hledání nejkratší cesty mezi dvěma uzly v daném grafu. Pod tímto pojmem si můžeme představit například hledání nejkratší cesty mezi danými dvě městy na mapě, nebo hledání nejlevnější cesty mezi dvěma uzly apod. V současnosti se využívají algoritmy, které fungují na principu složitějšího počítaní, než je zapotřebí. Nyní se seznámíme blíže s nejznámějšími algoritmy na hledání nejkratší cesty.

3.1 Dijkstrův algoritmus

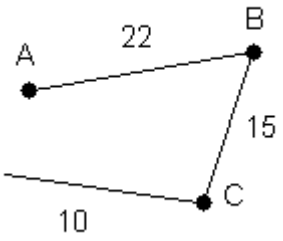
Dijkstrův algoritmus se aplikuje na graf G, ve kterém jsou hrany ohodnoceny pouze kladnými reálnými čísly. Hlavním principem je nalezení nejkratší cesty v grafu G od počátečního uzlu *p* ke koncovému uzlu *k*. V prvním kroku zvaném inicializace definujeme výchozí uzel, tzn. $D_0 = \{p\}$, u všech sousedních uzlů se nastaví hodnota dle ohodnocení hrany a u zbylých se nastaví hodnota ∞. V následujícím kroku hledáme nejmenší hodnotu hrany u sousedních uzlů. Jakmile uzel nalezneme vznikne nám množina, tzn. $D_1 = D_0 \cup \{a_1\}$, kde a_1 je uzel s momentálně nejnižší hodnotou při inicializaci. U všech sousedních uzlů se přepočte hodnota uzlů, tak že momentální hodnota se rovná hodnotě hrany z počátečního uzlu *p* do uzlu a_1, u všech zbylých uzlů nastavíme hodnotu opět ∞. V dalším kroku postupujeme obdobně, nalezneme sousední uzel s co nejnižší ohodnocenou hranou, čímž nám vznikne další množina $D_2 = D_1 \cup \{a_2\}$, přičemž a_2 je onen uzel s nejnižší hodnotou hrany. Provedeme další přepočet u sousedních uzlů, tím pádem bude momentální hodnota rovna délce od počátečního uzlu *p* do uzlu a_2, u všech zbylých uzlů nastavíme hodnotu ∞. Uvedeným způsobem

postupujeme dále, dokud nenastane situace, že uzel a_i se bude rovnat koncovému uzlu *k*. V tomto případě algoritmus končí a nejkratší cesta je nalezena.[3]

Důležitým pojmem u Dijkstrova algoritmu, který je nutno osvětlit, je pojem relaxace. Zjednodušeně můžeme říci, že je to zjištění, zda do daného uzlu neexistuje kratší cesta, než ta, co je tam uvedena doposud. Na obrázku č.6 můžeme vidět, že první byla stanovena cesta z uzlu A do uzlu B a její délka je 22, při dalším kroku vidíme, že přes uzel C do uzlu B je délka cesty 15, a proto vzniká nová kratší cesta. Obrácený postup, kdy je kratší cesta z uzlu A do B než přes uzel C do B vidíme na obrázku č.7.



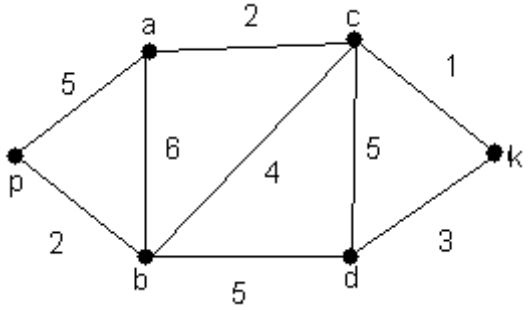
Obrázek č.6: Relaxace 1



Obrázek č.7: Relaxace 2

Příklad:

Představme si, že máme najít nejkratší cestu z v grafu G na obrázku č.8 mezi počátečním uzlem *p* a koncovým uzlem *k*.



Obrázek č.8: Graf (dijkstrův algoritmus)

[3] MEZNÍK, I. Diskrétní matematika pro užitou informatiku. 2009. s.53-57.

Postupovat budeme podle již zmíněných pravidel.

1. Inicializace: $D_0 = \{p\}$, sousední uzly jsou *a* a *b* a jejich délky jsou 5 a 2. U všech ostatních uzlů bude hodnota ∞.

2. Krok první: $D_1 = D_0 \cup \{b\} = \{p,b\}$ protože nejnižší hodnota při inicializaci patří uzlu *b*. Momentální hodnota bude tedy 2, jakožto vzdálenost mezi uzlem *p* a uzlem *b*. U sousedních uzlů *a,c,d* se provede přepočet nejkratší cesty vedoucí uzly *p,b*. Poté u zbylých uzlů nastavíme hodnotu ∞.

3. Krok druhý: $D_2 = D_1 \cup \{c\} = \{p,b,c\}$, neboť nejnižší hodnota z prvního kroku odpovídá uzlu *c*. U sousedních uzlů *a, d, k* se provede přepočet nejkratší cesty vedoucí uzly *p,b,c*. U všech ostatních bude hodnota ∞.

4. Krok třetí: $D_3 = D_2 \cup \{k\} = \{p,b,c,k\}$, neboť nejnižší hodnota z druhého kroku odpovídá uzlu *k*. V tomto příkladě vede nejkratší cesta v grafu G uzly *p,b,c,k* a její délka je 7.

Dijkstrův algoritmus lze implementovat jak na ohodnocený neorientovaný graf, tak na ohodnocený orientovaný graf. Jedná se o nejznámější algoritmus pro hledání nejkratší cesty. Jeho nevýhodou ovšem je, že pokud budou hrany obsahovat záporné hodnoty (zmíněno dále v podkapitole 3.3), algoritmus nebude správně fungovat.

Dijkstrův algoritmus se používá především při hledání cesty z bodu A do bodu B (např. Bludiště, internet routing atd.).

3.2 Floyd – Warshallův algoritmus

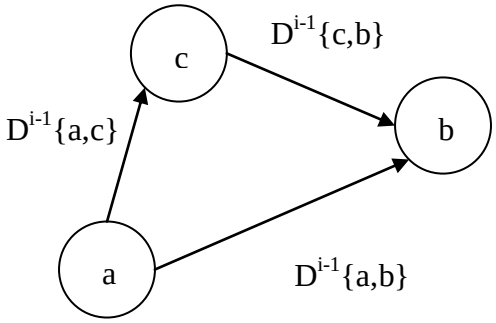
Se implementuje na orientovaném grafu, který neobsahuje záporné cykly. Najde délky nejkratších orientovaných cest mezi každou dvojicí uzlů. Navíc pokud nám nastane případ, že budeme mít stejné délky cest, vybere nám tu s nejnižším počtem hran.

Postup je následující - nejdříve si očíslujeme vrcholy čísly od 1 do n. Všechny dvojice vrcholů si nejlépe uložíme do matice n x n. Floyd – Warshallův algoritmus nepočítá vzdálenosti přímo, ale počítá je v n iteracích.

V i-té iteraci máme spočítanou matici D^i. Hodnota D^i {a,b} je hodnota délky nejkratší cesty z *a* do *b*, která smí procházet vrcholy {1,2,3…,i}. Jinými slovy můžeme říct, že D^i {a,b} je nejkratší cesta v podgrafu indukovaném vrcholy {1,2,3…,i}. V nulté iteraci začneme s maticí D^0 {a.b}, což je délka hrany mezi *ab*, pokud z *a* vede hrana do *b*, nula na diagonále a nekonečno jinak. V poslední iteraci skončíme s maticí D^n, která bude již obsahovat hledané vzdálenosti, neboť cesty přes *a* a *b* smí procházet všemi vrcholy.[4]

Principem algoritmu je, že projde všechny možnosti cest mezi každými dvěma vrcholy, a pokud zjistí, že cesta přes třetí vrchol by byla kratší, potom ji použije.

Vzdálenost(A,B) = min (Vzdálenost(A,B), Vzdálenost(A,C) + Vzdálenost(C,B))



Obrázek č.9: Graf (Floyd-Warshallův)

Výhodou tohoto algoritmu je jeho snadnější implementace než v případě Dijkstrova algoritmu a rychlost jeho naprogramování.

Nevýhodou je ovšem jeho výpočetní složitost. S rostoucím počtem vrcholů, mezi kterými vzdálenost počítáme, nám také roste počet kroků algoritmu.

[4] ČERNÝ, J. Základní grafové algoritmy [online]. 7.9.2008. s.88-89.

3.3 Bellman – Fordův algoritmus

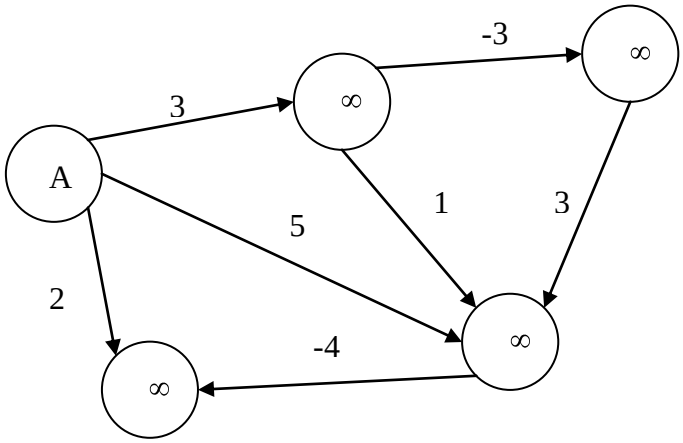
Řeší hledání nejkratší cesty v ohodnoceném grafu z jednoho uzlu do všech ostatních, s tím že hrany mohou být ohodnoceny i záporně. Představme si příklad, kdy dopravce projíždí určitou trasu a na trase dostává zaplaceno od odběratelů . Tyto platby budou mít v grafu podobu záporně ohodnocené hrany. Tudíž se náklady budou odečítat a snižovat.

Bellman –Fordův algoritmus jsou v podstatě dva do sebe vnořené cykly. První for cyklus probíhá $|U|$ - 1 tzn. počet uzlů mínus jedna a stanoví nám počet iterací. Druhý cyklus je vnořen do prvního a relaxuje všechny hrany. Čím více iterací proběhne, tím více se zpřesňují odhady nejkratších cest do uzlů. Jedná se o zobecnění Dijkstrova algoritmu.

Příklad:

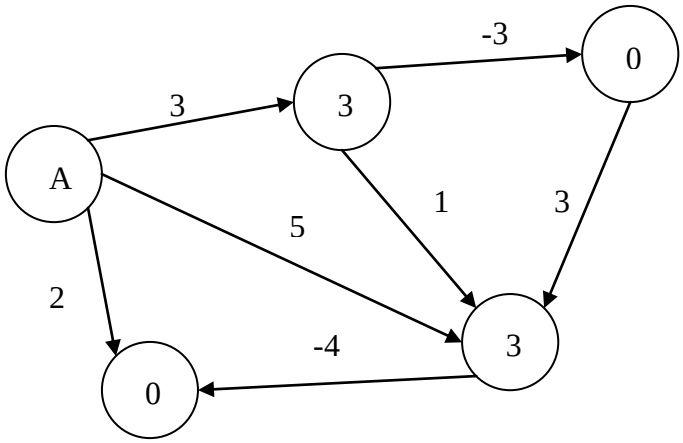
Podle uvedeného postupu nalezneme nejkratší cestu z uzlu A, za použití Bellman-Fordova algoritmu.

1. Inicializace : Probíhá stejně jako u Dijkstrova algoritmu. Počáteční uzel A je nastaven na hodnotu 0. Všechny ostatní jsou nastaveny na hodnotu ∞. Stanovíme si první (vnější) cyklus pomocí vzorce $|U|$ - 1 tzn. ($|5|$ - 1): 4 iterace. Druhý (vnitřní) cyklus ($|H|$): 7.



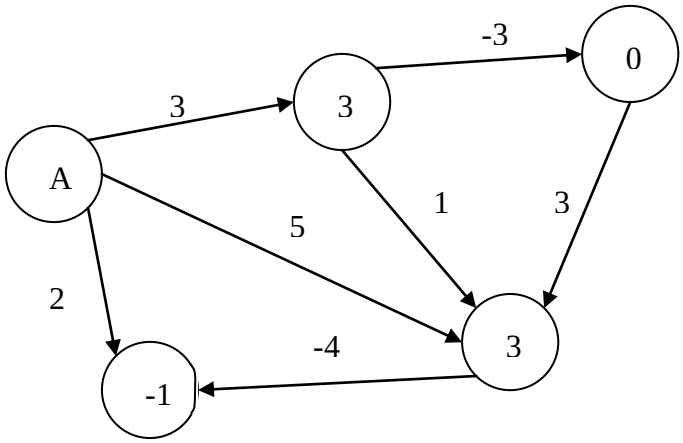
Obrázek č.10: Bellman-Ford 1

2. 1.iterace : V ní proběhne vnitřní cyklus tzn. projdou se všechny hrany grafu a změní se ohodnocení uzlů, za pomocí relaxace dojde k zapsání nejnižší hodnoty.



Obrázek č.11: Bellman-Ford 2

3. 2.iterace: V tomto kroku se provede cyklus znovu a dojde k opětovnému provedení relaxace.



Obrázek č.12: Bellman-Ford 3

4. 3. a 4. iterace: Ve třetí a čtvrté iteraci se nám již hodnoty uzlů měnit nebudou. Na obrázku č.12 můžeme vidět nalezení nejkratší cesty z uzlu A do uzlů ostatních, tímto zjištěním algoritmus končí.

Výhodou Bellman – Fordova algoritmu je, že můžeme na rozdíl od Dijkstry použít i záporně ohodnocené hrany. Další výhodou tohoto algoritmu je také fakt, že je schopen objevit záporně ohodnocené cykly. Princip funguje tak, že po doběhnutí již

zmíněných dvou cyklů vytvoříme další cyklus, který bude totožný s vnitřním cyklem, ovšem nebude již relaxovat. Pomocí tohoto cyklu pouze zjistíme, zda je možné relaxovat i nadále, potom to znamená, že algoritmus obsahuje záporné cykly a nebude fungovat.[5]

Bellman – Fordův algoritmus se využívá například ve směrovacím protokolu RIP. Tento protokol umožňuje routerům, komunikovat mezi sebou a reagovat na změny topologie počítačové sítě.

Nyní jsme se seznámili s algoritmy, které řeší cestu z počátečního vrcholu do koncového vrcholu. Můžeme říci, že následující dva typy algoritmů hledají nejkratší cestu mezi všemi vrcholy. Pro určité specifické úkoly, budeme muset použít následující algoritmy, které bývají označované jako hladové algoritmy. Obecně můžeme říci, že hladové algoritmy nám v každém kroku hledají lokální minimum, přičemž existuje šance, že nalezneme globální minimum.

[5] ČERNÝ, J. Základní grafové algoritmy [online]. 7.9.2008. s.89-91.

3.4 Algoritmy na hledání minimální kostry grafu

Než se začneme zabývat algoritmy na hledání minimální kostry grafu, ve stručnosti si uvedeme co to vlastně minimální kostra grafu je.

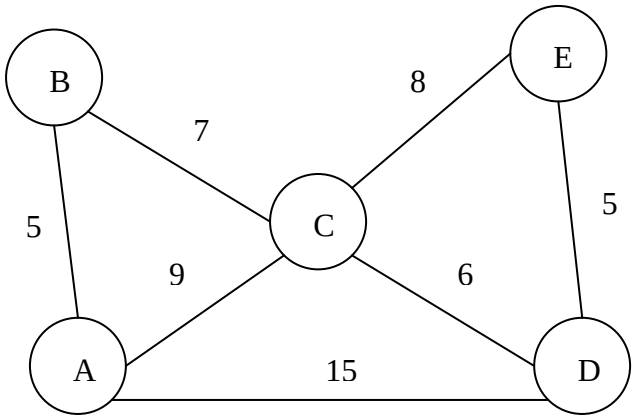
Pod pojmem kostra grafu si můžeme představit podgraf grafu, který obsahuje všechny uzly grafu a zároveň tolik hran, abychom byli schopni se dostat z libovolného uzlu do všech ostatních uzlů. Takových koster můžeme najít v jednom grafu i více, pokud není již samotný graf kostrou. Jiná situace ovšem nastane pokud budou hrany ohodnoceny nezápornou délkou. Potom můžeme najít minimální kostru grafu, to znamená kostru, která bude mít nejmenší součet ohodnocených hran.

3.4.1 Jarníkův algoritmus

Zajímavostí u tohoto algoritmu je, že v zahraničí je tento algoritmus znám pod názvem Primův algoritmus, který byl objeven v roce 1957. Oproti tomu Jarníkův algoritmus byl objeven již v roce 1930, tedy o 27 let dříve.[6]

Na jednoduchém grafu si vysvětlíme fungování tohoto algoritmu.

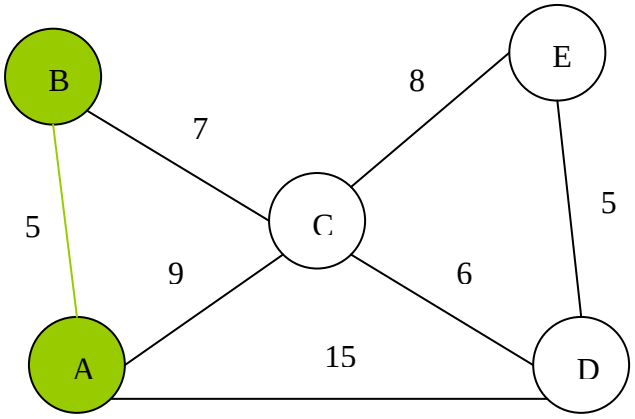
1. Můžeme vidět, že zvolený graf není strom, protože obsahuje kružnice. V prvním kroku si zvolíme výchozí vrchol - v našem případě bod A.



Obrázek č.13: Jarník 1

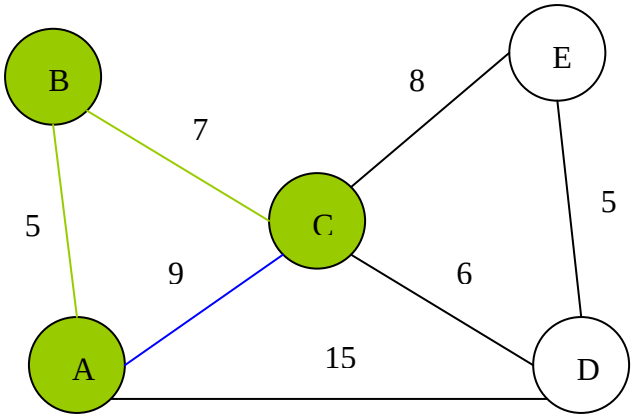
[6] NEŠETŘIL, J., MATOUŠEK, J. Kapitoly z diskrétní matematiky. 2003. s.161.

2. Nyní se budeme soustředit jen na vrcholy, do kterých vede přímo hrana z vrcholu A, a vybereme nejmenší hodnotu. Tzn. B = 5, C = 9, D = 15. Zvolíme nejmenší hodnotu, v tomto případě se jedná o vrchol B.



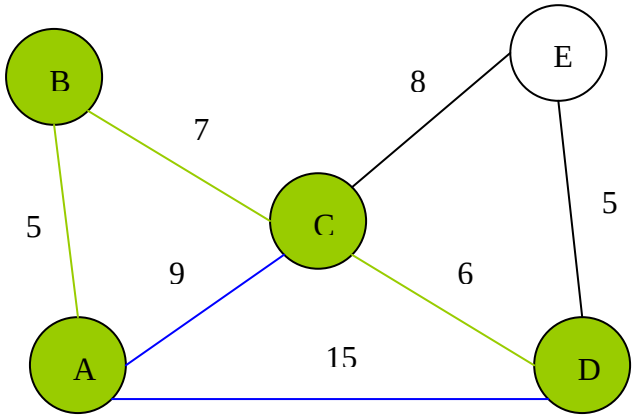
Obrázek č.14: Jarník 2

3. V tomto kroku se zaměříme na zbylé vrcholy a hrany vedoucí z vrcholu A a na vrcholy a hrany vedoucí z vrcholu B. Opět vybereme nejnižší hodnotu, s tím rozdílem, že hranu z A do B již použít nesmíme. Tedy hodnotu mezi vrcholy B a C. Zároveň můžeme odebrat hranu mezi A a C, jelikož by nám vznikla kružnice a graf by přestal být stromem a tudíž i kandidátem na kostru.



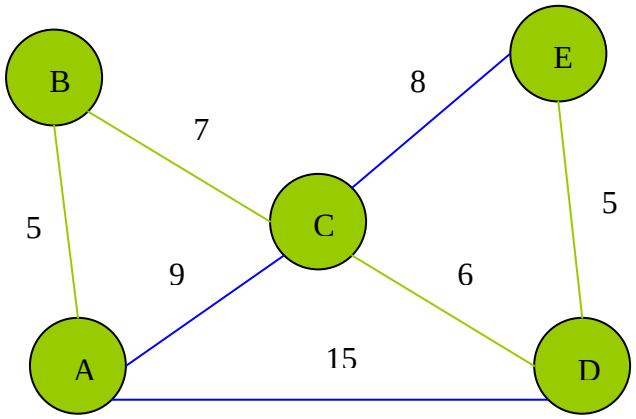
Obrázek č.15: Jarník 3

4. Stejným postupem pokračujeme i ve čtvrtém kroku. Nyní se zaměříme na vrchol A, ze kterého nám vede hrana pouze do D a na vrchol C, ze kterého nám vedou hrany do D a E. Vybíráme hranu mezi vrcholy C a D, jakožto nejnižší hodnotu a zároveň odstraníme hranu mezi A a D ,aby nám nevznikla kružnice.



Obrázek č.16: Jarník 4

5. V posledním kroku volíme nejkratší hranu jen mezi vrcholy D a E, nebo C a E. vybíráme kratší ohodnocení hrany a odstraňujeme poslední hranu, abychom nedostali v grafu kružnici.



Obrázek č. 17: Jarník 5

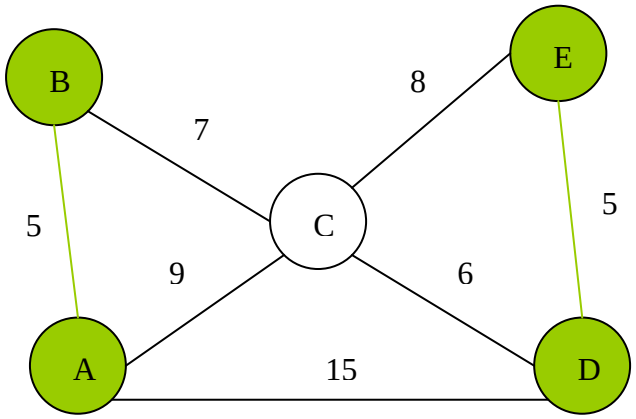
Z výsledného grafu můžeme vidět, že všechny vrcholy jsou již součástí stromu a tudíž jsme získali minimální kostru grafu. Její hodnota je 23.

3.4.2 Borůvkův algoritmus

Základním principem Borůvkova algoritmu je, že kostra vzniká postupně z hran, které mají ze všech možných hran spojujících dva uzly tu nejnižší hodnotu.

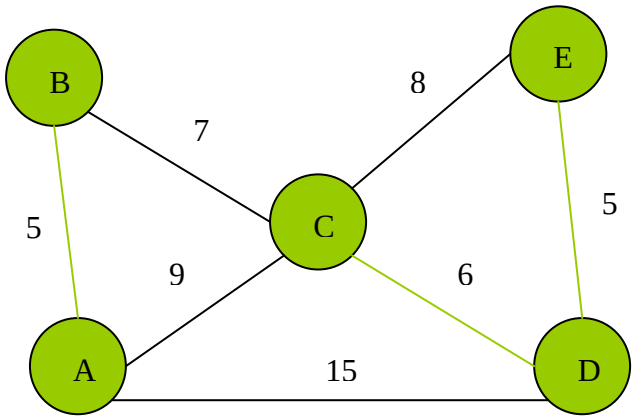
Abychom pochopili fungování algoritmu a zároveň také rozdíl mezi Jarníkovým a Borůvkovým algoritmem zvolíme pro náš příklad shodný graf.

1. V prvním kroku si z grafu vybereme nejnižší hodnotu hrany mezi dvěma uzly a označíme si ji. V našem případě se jedná o hodnotu 5, kterou mají hrany dvě, proto označíme první a posléze hned i druhou hranu..



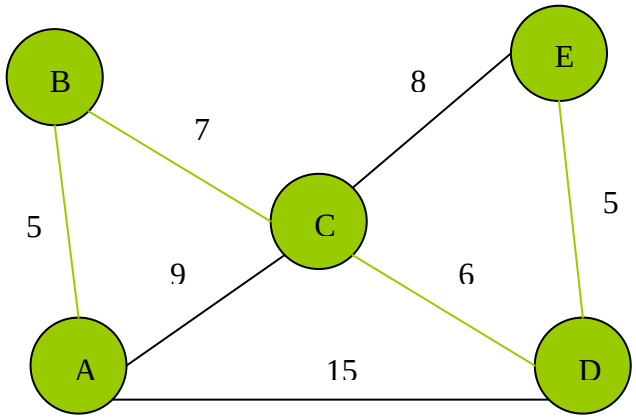
Obrázek č. 18: Borůvka 1

2. Nyní postupujeme dále a hledáme další doposud neoznačené hrany s nejnižší hodnotou. V našem grafu se jedná o hodnotu 6 mezi uzly C a D.



Obrázek č.19: Borůvka 2

3. V posledním kroku vybereme hodnotu 7 mezi uzly B a C. A tím dojde k propojení všech uzlů.



Obrázek č.20: Borůvka 3

Můžeme vidět, že jsme se dostali ke shodné minimální kostře grafu jako v případě Jarníkova algoritmu.

Borůvkův algoritmus byl využit pro elektrifikaci území. Uzly představují jednotlivé obce, hrany představují elektrické spojení a ohodnocení hran představuje cenu za natažení vedení mezi jednotlivými obcemi. Hlavní cíl byl nalezení takového spojení, aby byly všechny obce připojeny do elektrické sítě a přitom byla cena za spojení co nejnižší.[7]

Další úkoly, které mohou algoritmy na hledání minimální kostry grafu řešit jsou uvedeny v části praktické, konkrétně v podkapitole 6.4 Praktické využití programu.

V tuto chvíli jsme se seznámili se základními grafovými algoritmy po stránce funkčnosti jednotlivých algoritmů. Jak ovšem vybrat vhodný algoritmus pro situaci, která má být řešena? Na tuto otázku nám odpoví následující kapitola.

[7] NEŠETŘIL, J., MATOUŠEK, J. Kapitoly z diskrétní matematiky. 2003. s.161-165.

Porovnávání algoritmů

Pokud chceme porovnávat jednotlivé algoritmy mezi sebou, můžeme se na tento problém dívat z více úhlů. Chceme, aby algoritmus proběhl co nejrychleji, nebo chceme aby využíval méně paměti, či snad aby programování zdrojového kódu bylo co nejjednodušší? Všechny otázky jsou důležité a záleží jen na tom, na který problém chceme daný algoritmus implementovat.

- Rychlost výpočtu – udává dobu, za kterou algoritmus proběhne, a my se dobereme k požadovanému výsledku.
- Paměťová náročnost – udává, kolik paměti zabere samotný program. Pokud bude paměti málo, použijeme paměť z pevného disku, čímž se ovšem zpomalí rychlost výpočtu.
- Rychlost za jakou napíšeme program – udává dobu, za kterou vytvoříme zdrojový kód programu. Spadá sem také odstraňování chyb vzniklých při programování. U složitých programů uděláme chybu spíše než u programů s kratším a jednodušším zdrojovým kódem.

Každý správný programátor si na začátku programování stanový na které z daných kritérií se zaměří a který algoritmus bude pro jeho práci nejefektivnější.

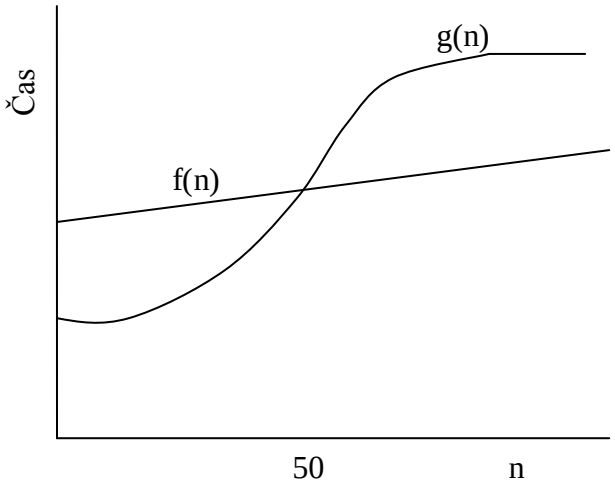
4.1 Časová složitost a prostorová složitost

Kritérium rychlost výpočtu nás přivádí k důležitému pojmu, a to k časové složitosti. Zjednodušeně nám časová složitost říká, jak dlouho program poběží v závislosti na velikosti vstupních dat.

S časovou složitostí úzce souvisí prostorová (paměťová) složitost, která nám říká, kolik potřebujeme paměti k vykonání určitého algoritmu v závislosti na velikosti vstupní dat.

Typickým příkladem snížení časové složitosti je že si něco před počítáme. Faktem ovšem zůstává, že před počítané údaje si musíme někam uložit, čímž se nám zvyšuje

prostorová složitost. Proto často dochází k tomu, že rychlejší algoritmy mají větší prostorovou složitost a obráceně. Pokud máme dobře změřeno chování algoritmů (časovou složitost) na různě velkých datech, můžeme začít porovnávat. Některý algoritmus může být lepší pro menší data (n<50), jiný naopak pro větší data (n>50).



Obrázek č.21: Porovnání algoritmů

Pokud dojde k této situaci je ideální vybrat kombinaci obou dvou algoritmů. Pro data malá použijeme první algoritmus a pro data velká algoritmus druhý.

U všech zmíněných algoritmů se používá asymptotická časová složitost, která zjišťuje, jakým způsobem se bude měnit chování algoritmu v závislosti na velkosti vstupních dat.

Nyní se blíže seznámíme s časovou složitostí algoritmů uvedených v této práci.

- Dijkstrův, Jarníkův a Borůvkův algoritmus – Časová složitost je $O(|H| \cdot \log |U|)$.
- Floyd warshallův algoritmus – má časovou složitost $O(|U|^3)$.
- Bellman fordův algoritmus – má časovou složitost $O(|U| \cdot |H|)$.

Pokud budeme jako hlavní kritérium brát dobu naprogramování, je nejlepším východiskem Floyd warshallův algoritmus. Naopak pokud budeme brát jako hlavní

kritérium složitost výpočtu, potom využijeme Dijkstrův algoritmus. Další variantou může být odhalení záporných cyklů v grafu, v tomto případě se implementuje Bellman fordův algoritmus. Musíme si uvědomit, že každý z uvedených algoritmů se implementuje na různé druhy úloh, a proto se algoritmy používají podle konkrétní situace.[8]

Nyní jsme se seznámili s fungováním a porovnáváním jednotlivých algoritmů a nic nám nebrání v tom, abychom se mohli přesunout k části praktické uvedené v následující části.

[8] ČERNÝ, J. Základní grafové algoritmy [online]. 7.9.2008. s.1-7.

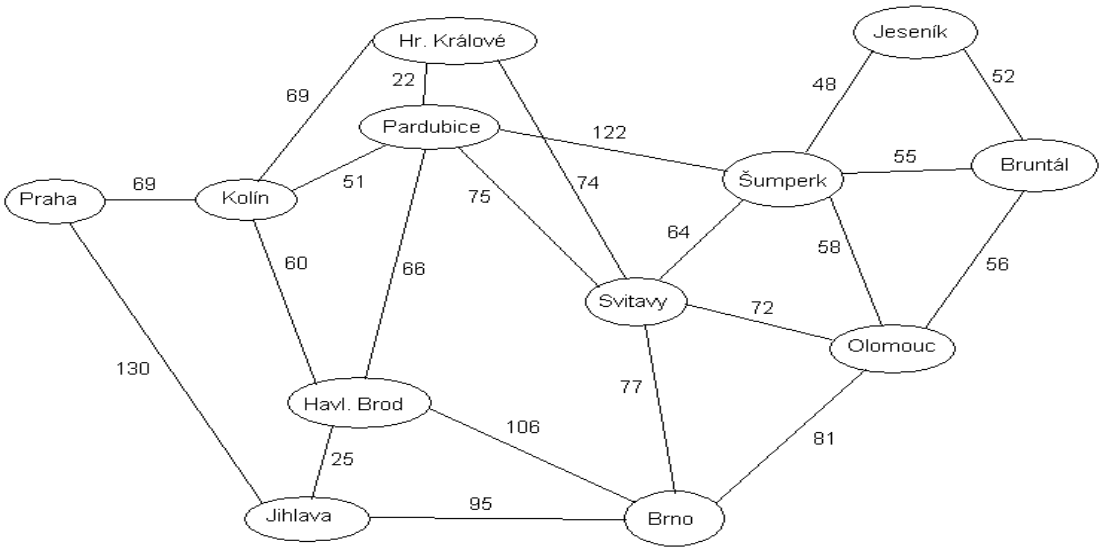
Praktická část

5 Úvod

V praktické části se zaměříme na vytvoření algoritmu, na hledání minimální kostry grafu. Vzhledem k tomu, že většina GPS navigací, vyhledávačů nejkratších cest v mapách a obdobných programů využívá algoritmy sestavené na základě Dijkstrova či Floyd-Warshallova algoritmu. Zaměříme se na oblast vyhledávání minimální kostry grafu, která se neobjevuje až tak často oproti zmíněným algoritmům, ovšem má také široké uplatnění. Vytvořený algoritmus v programu MS Excel 2003 s využitím jazyka VBA popíšeme v jednotlivých bodech a shrneme si využití celého programu v praktických příkladech.

5.1 Vstupní údaje

Pro lepší porozumění navrhneme model, kterým se budeme v praktické části zabývat. Vezměme si případ, kdy budeme chtít zapojit telefonní síť v části ČR tak, aby byly naše náklady na zapojení co nejnižší. Na obrázku č.22 můžeme vidět souvislý graf měst, včetně vzdáleností mezi jednotlivými městy.

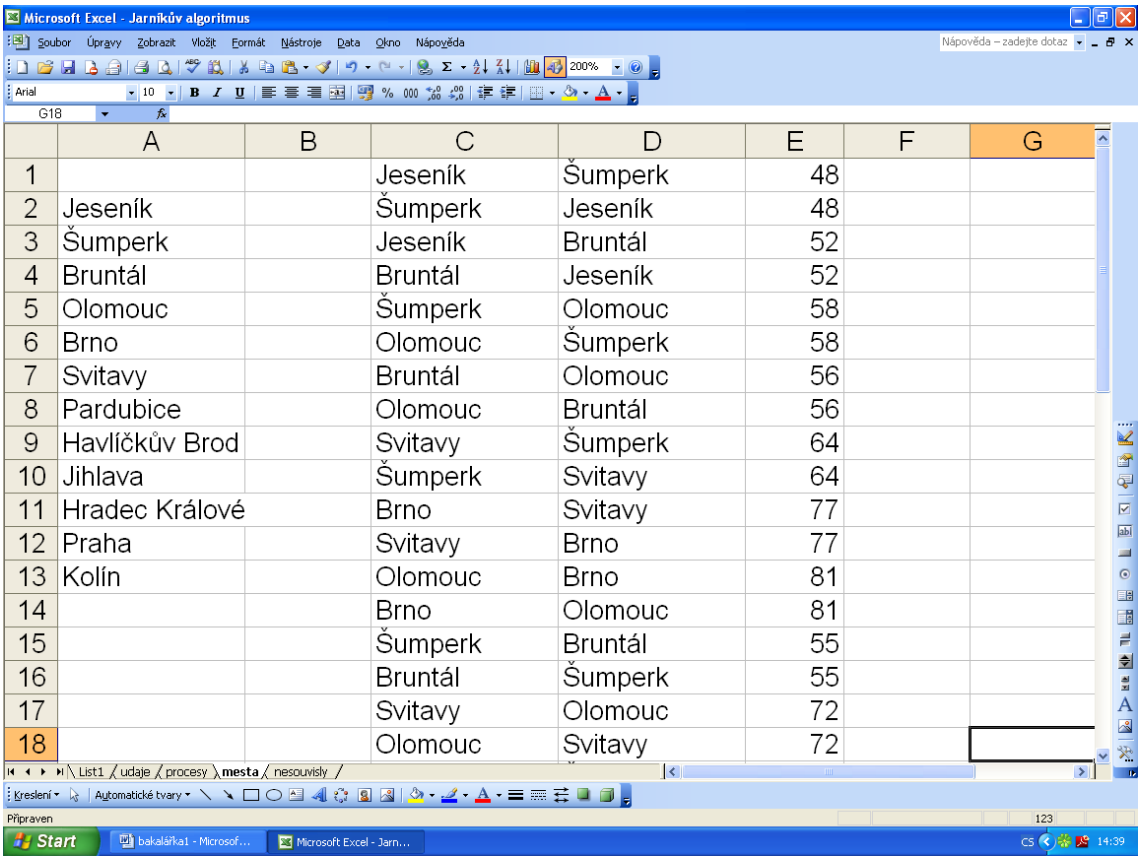


Obrázek č.22: Mapa

6 Vlastní návrh řešení

6.1 Základní nastavení

Pro samotné zjištění minimální kostry grafu využijeme Jarníkův algoritmus. V prvním kroku, v programu Excel, vytvoříme nový list, který pojmenujeme např. *mesta*. Na tento list vypíšeme údaje získané z grafu způsobem zobrazeným na obrázku č.23.



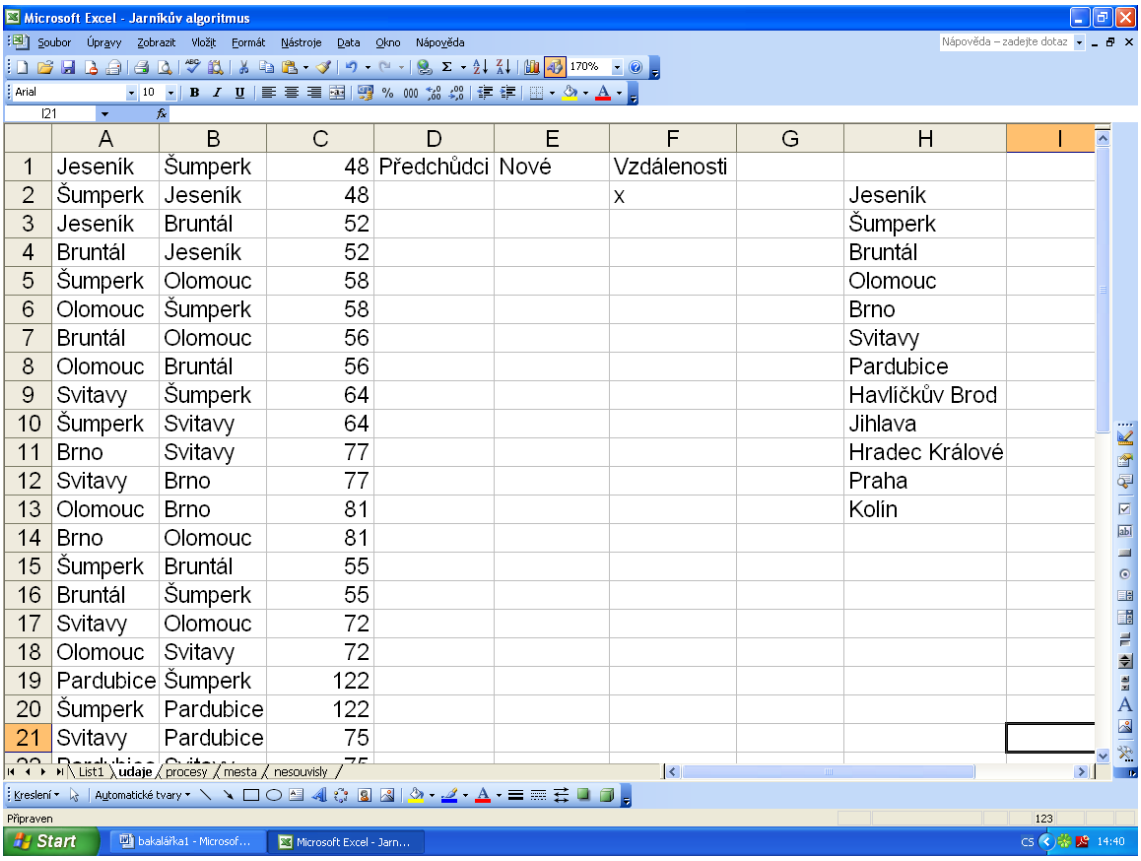
	A	B	C	D	E	F	G
1			Jeseník	Šumperk	48		
2	Jeseník		Šumperk	Jeseník	48		
3	Šumperk		Jeseník	Bruntál	52		
4	Bruntál		Bruntál	Jeseník	52		
5	Olomouc		Šumperk	Olomouc	58		
6	Brno		Olomouc	Šumperk	58		
7	Svitavy		Bruntál	Olomouc	56		
8	Pardubice		Olomouc	Bruntál	56		
9	Havlíčkův Brod		Svitavy	Šumperk	64		
10	Jihlava		Šumperk	Svitavy	64		
11	Hradec Králové		Brno	Svitavy	77		
12	Praha		Svitavy	Brno	77		
13	Kolín		Olomouc	Brno	81		
14			Brno	Olomouc	81		
15			Šumperk	Bruntál	55		
16			Bruntál	Šumperk	55		
17			Svitavy	Olomouc	72		
18			Olomouc	Svitavy	72		

Obrázek č.23: List mesta

Jak můžeme vidět z obrázku, v prvním sloupci A jsou uvedeny všechna města, která se v daném grafu vyskytují. V ostatních sloupcích máme vypsány všechny vzdálenosti mezi danými dvěma městy, včetně vzdáleností mezi městy prohozenými (tzn.města v opačném pořadí). Umístění údajů je čistě náhodné a nemá žádný hlubší význam, dotýká se až údajů obsažených v samotném zdrojovém kódu programu.

V dalším kroku vytvoříme pracovní list, který můžeme nazvat např. *udaje*. S tímto listem budeme pracovat nejvíce, neboť na něm budou probíhat veškeré výpočty, vedoucí až k samotnému nalezení minimální kostry grafu.

Program funguje tak, že při otevření souboru se zobrazí nabídka, se kterou částí chceme pracovat, v našem případě se jedná o města. Další variantou můžou být například firemní procesy, či jiný konkrétní příklad, který si stanovíme. Po kliknutí na možnost města se v listu *udaje* zobrazí počáteční stav vyobrazený na následujícím obrázku.



	A	B	C	D	E	F	G	H	I
1	Jeseník	Šumperk	48	Předchůdci	Nové	Vzdálenosti			
2	Šumperk	Jeseník	48			x		Jeseník	
3	Jeseník	Bruntál	52					Šumperk	
4	Bruntál	Jeseník	52					Bruntál	
5	Šumperk	Olomouc	58					Olomouc	
6	Olomouc	Šumperk	58					Brno	
7	Bruntál	Olomouc	56					Svitavy	
8	Olomouc	Bruntál	56					Pardubice	
9	Svitavy	Šumperk	64					Havlíčkův Brod	
10	Šumperk	Svitavy	64					Jihlava	
11	Brno	Svitavy	77					Hradec Králové	
12	Svitavy	Brno	77					Praha	
13	Olomouc	Brno	81					Kolín	
14	Brno	Olomouc	81						
15	Šumperk	Bruntál	55						
16	Bruntál	Šumperk	55						
17	Svitavy	Olomouc	72						
18	Olomouc	Svitavy	72						
19	Pardubice	Šumperk	122						
20	Šumperk	Pardubice	122						
21	Svitavy	Pardubice	75						

Obrázek č.24: List udaje

Jak můžeme z obrázku vidět, vypsali se všechna města a všechny cesty, včetně vzdáleností vždy mezi právě dvěma městy. Na začátku vyhledávání si uživatel ve formuláři zvolí, které bude počáteční město, od kterého se bude vyhledávání odvíjet. Pokud bychom neuvažovali daný příklad, který se týká rozvedení telefonní sítě, ale

například firemních procesů, potom bychom měli na listu *udaje* místo měst právě zmíněné procesy. Záleží na daném příkladu (zmíněno v podkapitole 6.4).

6.2 Popis zdrojového kódu a průběhu programu

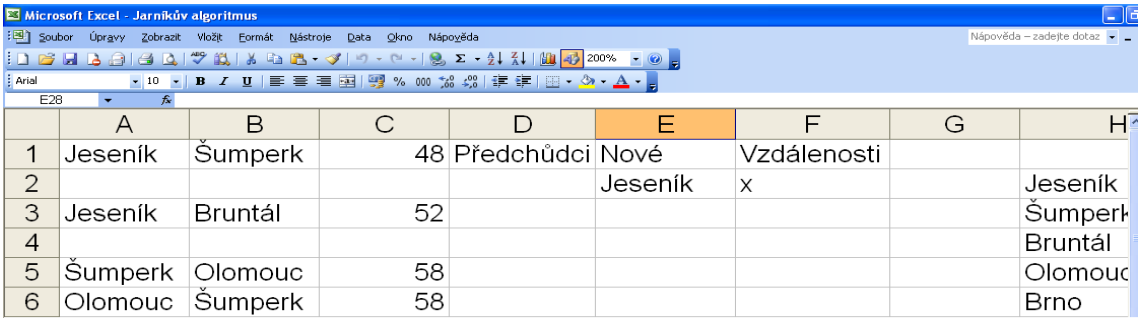
Zdrojový kód tvoří hlavní srdce celého programu. V této podkapitole dojde k popisu postupu vytvoření jednotlivých částí zdrojového kódu spolu s detailnějším rozborem a popisem hlavní části programu.

Na začátku programu si musí uživatel zvolit jednu z vyhledávaných oblastí, v našem případě se jedná o města. Pokud tak neučiní, zobrazí se mu upozornění, že pokud chce vyhledávat minimální kostru grafu, je nutné zvolit jednu z oblastí, která je v nabídce programu. K řešení této situace je využit jednoduchý cyklus. Následně využijeme další cyklus, který spočítá celkový počet vrcholů (měst) v grafu. Význam spočítání všech vrcholů spočívá v tom, že pokud budeme mít navštíveno v naší kostře grafu stejný počet vrcholů jako je jejich celkový počet v grafu, máme jistotu, že jsme nalezli minimální kostru grafu a zároveň se jedná o souvislý graf.

V další fázi programu si uložíme počáteční město (v našem případě Jeseník) do proměnné *aktuální:*

aktualni = ComboBox1.Value

Nyní program nalezne ze sloupce označeného B všechny města, která se s počátečním městem shodují, a vymaže celý příslušný řádek včetně vzdálenosti, kterou už nebudeme potřebovat. Důvodem je zajištění nemožnosti vytvoření kružnice a zabránění tak nenalezení minimální kostry grafu. Tuto situaci můžeme vidět na obrázku č. 25.



	A	B	C	D	E	F	G	H
1	Jeseník	Šumperk	48	Předchůdci	Nové	Vzdálenosti		
2					Jeseník	x		Jeseník
3	Jeseník	Bruntál	52					Šumperk
4								Bruntál
5	Šumperk	Olomouc	58					Olomouc
6	Olomouc	Šumperk	58					Brno

Obrázek č.25: Počáteční město

V další fázi programu již přistupujeme k vytvoření samotného cyklu, který poběží do té doby, dokud nenajde minimální kostru grafu, proto si ho popíšeme důkladněji včetně ukázek zdrojového kódu.

V prvním kroku si musíme nastavit hodnoty jednotlivých proměnných, které se budou či nebudou v programu měnit.

```
pozice = 2
citac = 0
nejmensi = 999
pocet = 0
```

Obrázek č.26: Nastavení proměnných

K osvětlení zadání právě uvedených hodnot se dostaneme v dalších krocích. V následujícím kroku si vytvoříme jednoduchý cyklus, který spočítá počet měst uvedených ve sloupci E, v současné době se jedná o město pouze jedno, a to Jeseník. Do proměnné *pocet*, se v prvním běhu cyklu tedy uloží číslo jedna.

Poté proběhne cyklus uvedený na následujícím obrázku.

```
pokracovani1:

For i = pozice To 50
If Worksheets("udaje").Range("E" & i).Value <> "" Then
   hodnota = Worksheets("udaje").Range("E" & i).Value
   pozice = i + 1
   citac = citac + 1
   GoTo pokracovani
   Exit For

End If
Next i
```

Obrázek č.27: Cyklus1

Význam slova *pokracovani1* bude osvětlen dále, neboť v programu se můžeme vracet, pokud nebude splněna některá z podmínek, nebo naopak bude pomocí klíčového slova GoTo. Cyklus začíná běžet od čísla dvě, jelikož proměnná *pozice* má nastavenou hodnotu dva. Zkontroluje se tedy podmínka If, a protože není druhý řádek ve sloupci E

prázdný, do proměnné *hodnota* se uloží právě Jeseník, *pozice* se navýší o jedna, tedy na hodnotu tři a *citac* se změní z hodnoty nula na hodnotu jedna. Poté se přesuneme pomocí GoTo pokracovani na další příslušnou pozici ve zdrojovém kódu, kterou můžeme vidět na následujícím obrázku.

```
pokracovani:

For i = 1 To 50
If Worksheets("udaje").Range("A" & i).Value = hodnota And
            Worksheets("udaje").Range("C" & i).Value < nejmensi Then
  nejmensi = Worksheets("udaje").Range("C" & i).Value
  nove = Worksheets("udaje").Range("B" & i).Value
  aktualni = Worksheets("udaje").Range("A" & i).Value

End If
Next i
```

Obrázek č.28: Cyklus2

Dostáváme se do dalšího cyklu, nyní budeme hledat ve sloupci A město, se kterým aktuálně pracujeme, to znamená Jeseník a zároveň musí být vzdálenost z Jeseníku do jakéhokoliv dalšího města nižší, než je hodnota proměnné *nejmensi*. Ve sloupci A může být aktuální město vícekrát, a proto máme jistotu, že nalezneme právě to, které má nejnižší hodnotu vzhledem k tomu, že se při každém objevení aktuálního města vzdálenosti porovnají. Do proměnné *nejmensi* se uloží nejnižší vzdálenost, do proměnné *nove* se uloží město které leží na stejném řádku ve sloupci B jako aktuální město ve sloupci A.

Poté, co cyklus proběhne, přichází na řadu porovnání, zda-li jsme prošli všechny města ve sloupci E. Zde přichází na scénu cyklus, který jsme provedli na začátku programu a který spočítal právě počet měst ve sloupci E a uložil je do proměnné *pocet* a také *citac*, který proběhl v cyklu na obrázku č.27.

```
If pocet <> citac
GoTo pokracovani1
End If
```

Obrázek č.29: Porovnání1

V našem případě na nás funkce If nebude mít vliv, jelikož se proměnná *pocet* rovná proměnné *citac* (obě mají hodnotu jedna), proto může program běžet dál a nemusí se vracet zpět k prvnímu cyklu.

Následuje zaznamenání nových údajů do sloupce *Předchůdci*, kde se zapíše předchozí město, a do sloupce *Nové*, kde se zapíše město nové. Dále se také zaznamená vzdálenost mezi dvěma danými městy ve sloupci *Vzdálenosti*. Jakmile máme tyto údaje zaznamenány, nebudeme již město označené jako *nove* potřebovat ve sloupci B, proto jej nalezneme a celý řádek opět vymažeme. V poslední fázi již navýšíme proměnnou *citac* o hodnotu jedna, neboť jsme našli právě jedno nové město, proto bude jeho současná hodnota rovna dvěma. Spustíme poslední cyklus na ověření souvislosti grafu. Graf je nesouvislý, pokud se ve sloupci E objeví hodnota 999, jakožto počáteční hodnota proměnné *nejmensi*. V našem případě by to znamenalo, že z města Jeseník nevede žádná cesta.

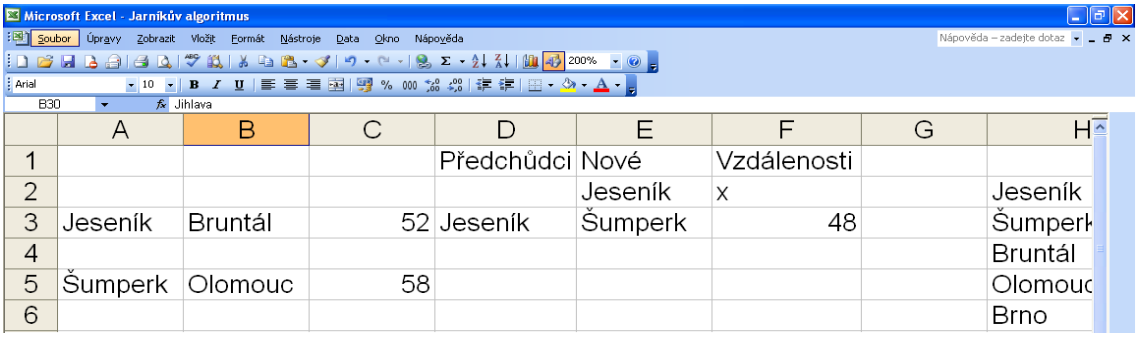
Poté ověříme podmínku if, která je uvedena na obrázku číslo 30.

```
If citac <> vychozi Then
GoTo pokracovani2
End If
```

Obrázek č.30: Porovnání2

Podmínka je splněná, neboť hodnota *citac* se rovná dvěma, a *hodnota*, která představuje celkový počet měst se rovná dvanácti, prakticky to znamená, že jsme zatím navštívili pouze dvě města, není proto možné, aby byla nalezena minimální kostra grafu, proto se vracíme zpět až před nastavení prvních proměnných na obrázku č.26.

Nyní si popíšeme kroky, které se oproti prvnímu proběhnutí hlavní části programu změní. Zásadní rozdíl můžeme vidět na obrázku číslo 31, který se nachází na následující straně.

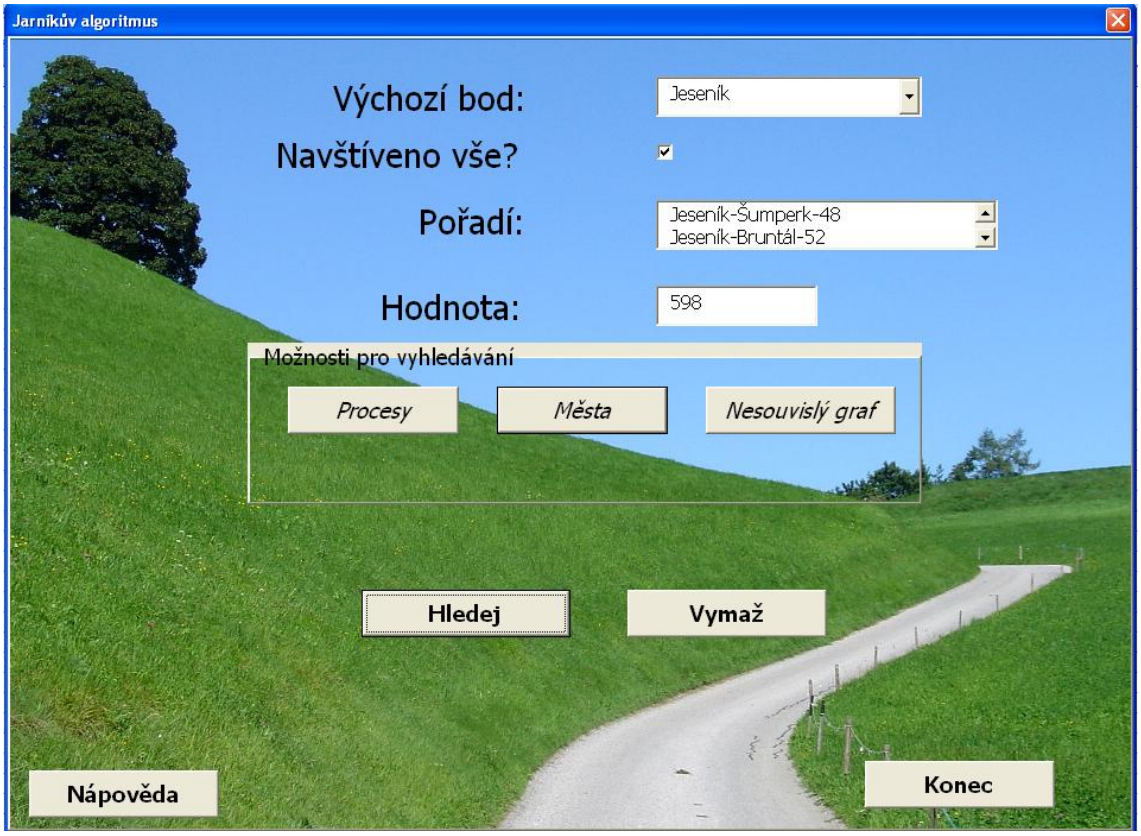


	A	B	C	D	E	F	G	H
1				Předchůdci	Nové	Vzdálenosti		
2					Jeseník	x		Jeseník
3	Jeseník	Bruntál	52	Jeseník	Šumperk	48		Šumperk
4								Bruntál
5	Šumperk	Olomouc	58					Olomouc
6								Brno

Obrázek č.31: Druhé město

V této chvíli nastane v programu jeden zásadní rozdíl oproti prvnímu proběhnutí daných cyklů. Jakmile se v programu dostaneme k podmínce If, uvedené na obrázku číslo 29, zjistíme, že podmínka je splněna, neboť ve sloupci E se v současné době nachází dvě města, tudíž proměnná *pocet* nabude hodnoty dva. Ovšem proměnná *citac* bude mít hodnotu jedna, neboť prozatím jsme zkontrolovali pouze cesty vedoucí z města Jeseník. Proto se vrátíme k cyklu na obrázku číslo 27 pomocí klíčového slova GoTo a k již porovnaným cestám z Jeseníku, porovnáme také cesty ze Šumperka. V této chvíli budeme mít jistotu, že jsme nalezli nejkratší cestu vedoucí buď z Jeseníku, nebo ze Šumperka do města nového, a zároveň, že hodnota proměnné *citac* je rovna dvěma, a tudíž nás program pustí dále. Program již zapíše do sloupce *Předchůdci*, o které ze dvou uvedených měst se jedná, dále do sloupce *Nové* město, do kterého jsme se dostali, a vzdálenost mezi těmito městy do sloupce *Vzdálenosti*.

Ukončení běhu programu nastane ve chvíli, kdy budeme mít ve sloupci *Nové* stejný počet měst jako ve sloupci *Města*. Minimální kostra grafu je nalezena a my můžeme vidět výsledek vyhledávání na obrázku číslo 32, který je uveden na následující straně. Zde si také provedeme rozbor jednotlivých ovládacích prvků programu, abychom se mohli lépe zorientovat a pochopit tak plně fungování celého programu.

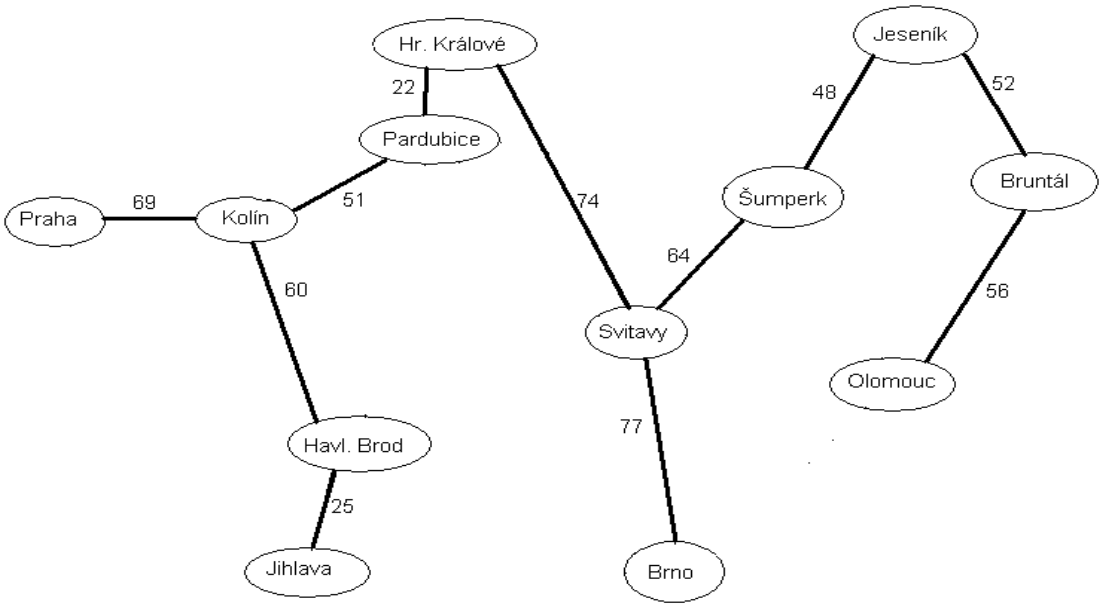


Obrázek č.32: Výstup programu

V kolonce *Výchozí bod* můžeme vidět počáteční město, ze kterého jsme vycházeli. Pokud je graf souvislý a byla navštívena všechna města, pro kontrolu se odškrtne políčko *Navštíveno vše?*. Dále vidíme kolonku *Pořadí*. Zde máme postupně seřazeny jednotlivé dvojice měst a vzdálenosti mezi nimi, abychom viděli, jak program postupoval. Poslední kolonka *Hodnota* udává vzdálenost v km nalezené minimální kostry grafu. U toho posledního údaje neuvádíme pevnou jednotku, o kterou se jedná, neboť se zde můžou měnit jednotky vzdálenosti, jednotky času, ceny a podobně.

Pokud se uživatel programu ztratí v ovládání stačí, když si klikne na tlačítko *Nápověda*, kde se dozví veškeré informace a postupy, od významu jednotlivých tlačítek až po řešení situace nesouvislého grafu, která je detailněji popsána v následující kapitole Nesouvislý graf. Tlačítko *Konec* plní funkci ukončení programu.

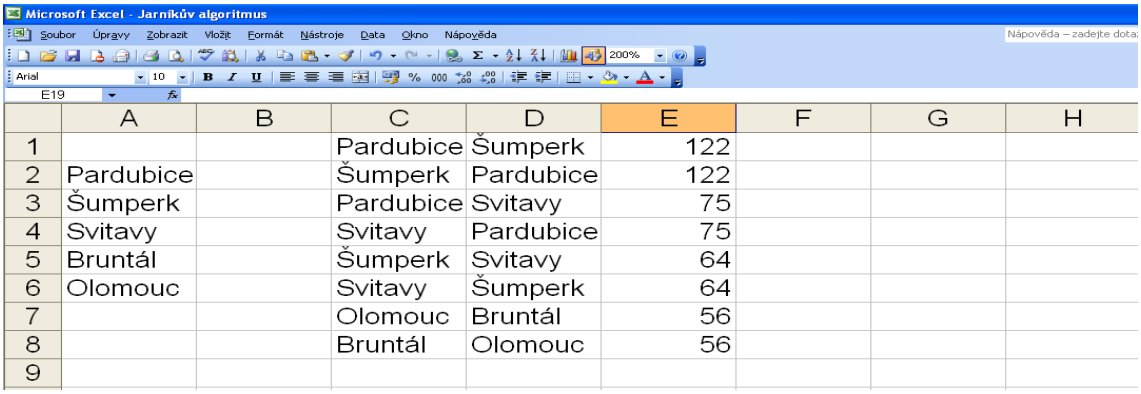
Následující obrázek nám graficky znázorňuje výsledek programu.



Obrázek č.33: Grafické znázornění

6.3 Nesouvislý graf

Nyní si blíže popíšeme případ kdy program minimální kostru grafu nenalezne. To znamená, že graf není souvislý. Tuto možnost si ukážeme na krátkém příkladu s možným řešením dané situace. Pro tento případ si navrhneme nový graf, který si nejdříve ukážeme zapsaný v programu na obrázku číslo 34.

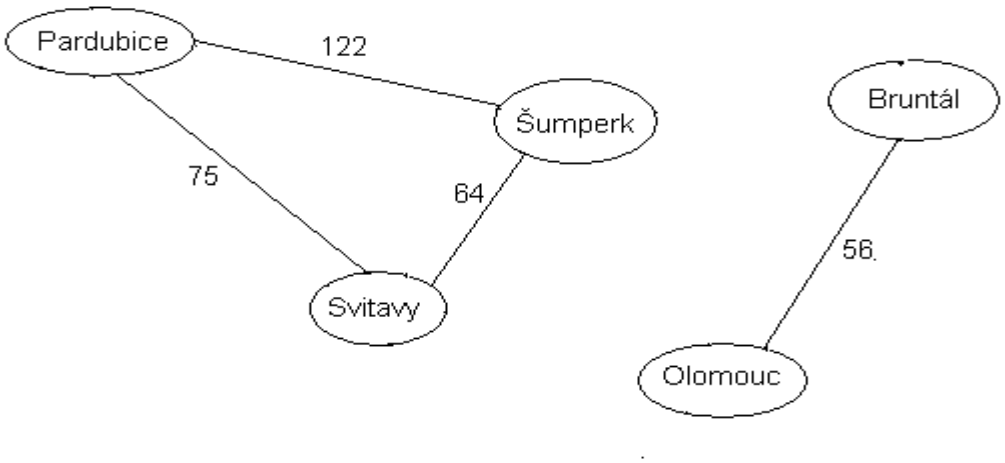

Microsoft Excel - Jarníkův algoritmus

	A	B	C	D	E	F	G	H
1			Pardubice	Šumperk	122			
2	Pardubice		Šumperk	Pardubice	122			
3	Šumperk		Pardubice	Svitavy	75			
4	Svitavy		Svitavy	Pardubice	75			
5	Bruntál		Šumperk	Svitavy	64			
6	Olomouc		Svitavy	Šumperk	64			
7			Olomouc	Bruntál	56			
8			Bruntál	Olomouc	56			
9								

Obrázek č.34: List nesouvisly

Na první pohled toho zápis moc neřekne, zkusme si představit, že nebudeme mít v grafu pouze pět měst jako je tomu v tomhle případě, ale měst budeme mít například padesát. Obecně můžeme říci, že čím více měst budeme mít, tím horší pro nás bude možnost rozpoznání, zda je graf souvislý, či ne, a především potom rozpoznání, která města lze v daném grafu propojit a která ne.

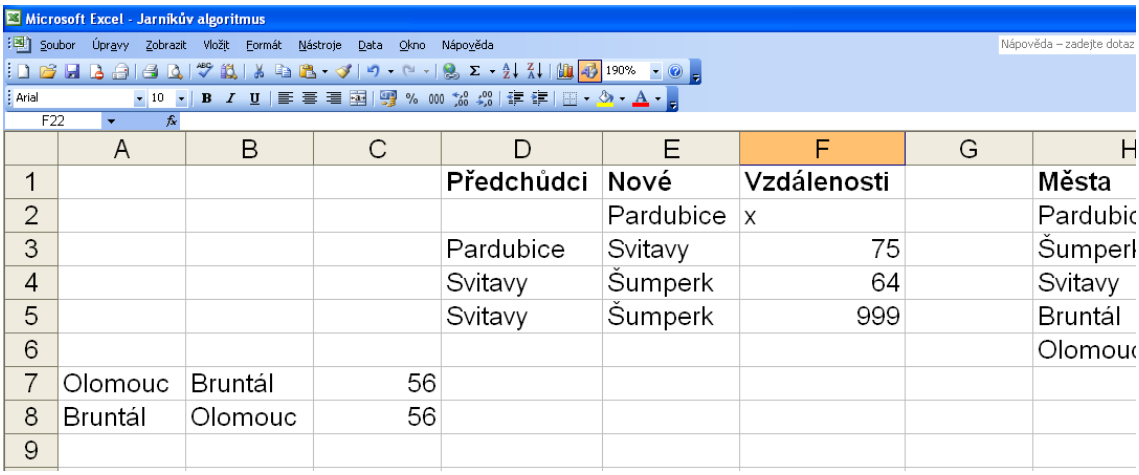
Abychom si dokázali uvedené údaje na obrázku číslo 34 lépe představit, podíváme se na následující obrázek číslo 35, kde máme grafické znázornění daných údajů.



Obrázek č.35: Nesouvislý graf

Nyní, když jsme schopni si blíže představit, jak uvedený nesouvislý graf vypadá, můžeme se přesunout k části, kde uvidíme, jak bude program reagovat na zjištění, že graf je nesouvislý. Důležité je, uvědomit si fakt, že vytvořené listy, ze kterých bereme údaje, mohou být vytvořeny několika způsoby. Lze je nakopírovat například z internetu, jakožto údaje o naší firmě, se kterými chceme pracovat. Další možností je například vypsání údajů ručně některým ze zaměstnanců. Může ovšem nastat situace, kdy budou údaje na internetu či v interní databázi firmy špatně zaznamenány či dojde k selhání lidského faktoru. Je nutné, aby byl program schopen reagovat na tuto skutečnost, jinak by mohly vznikat chybné výsledky, které by mohly ohrozit činnosti navazující na výsledky programu.

Jako výchozí bod si zvolíme například město Pardubice. Spustíme vyhledávání a objeví se chybné okno, které říká, že je graf nesouvislý. Všechny kolonky na formuláři zůstanou prázdné. Poté musí uživatel zavřít program a přesunout se na příslušný list. V našem případě na list nazvaný *udaje*. Vzhled listu můžeme vidět na obrázku číslo 36.



	A	B	C	D	E	F	G	H
1				**Předchůdci**	**Nové**	**Vzdálenosti**		**Města**
2					Pardubice	x		Pardubic
3				Pardubice	Svitavy	75		Šumperk
4				Svitavy	Šumperk	64		Svitavy
5				Svitavy	Šumperk	999		Bruntál
6								Olomouc
7	Olomouc	Bruntál	56					
8	Bruntál	Olomouc	56					
9								

Obrázek č.36: List udaje výpis měst

Zde již můžeme jednoduše zjistit, která města propojena jsou a která ne. Ve sloupci *Nové* můžeme vidět tři města Pardubice, Svitavy a Šumperk. To jsou města, která doposud graf tvořily. Ve sloupci *Vzdálenosti* však můžeme vidět číslo 999, které v našem případě představuje nekonečno a programu říká, že doposud nebyli navštíveny všechna města v grafu. Ve sloupci A můžeme vidět města, která nejsou se zbylými městy propojena, a to znamená, že graf není souvislý. Je nutné, aby uživatel zkontroloval údaje, se kterými pracoval. V našem případě se jedná o města a znamená to, že se jedná spíše o chybu v zápisu, nežli aby to byla skutečně pravda, a města spolu neměla žádnou vazbu. Proto se uživatel jednoduše vrátí na list *nesouvisly* a údaje poopraví podle skutečnosti. Druhou variantou může být fakt, že uvedený graf není možné změnit na graf souvislý, protože skutečně není možné vytvořit ze získaných údajů žádnou vazbu mezi všemi vrcholy, a potom není možné v tomto grafu nalézt minimální kostru grafu.

6.4 Praktické využití programu

Na samém začátku shrnutí praktického využití programu vyzvedneme fakt, že uživatel (firma), nebude muset složitě instalovat nový software, na kterém mu bude program fungovat, jelikož v současné době se téměř ve všech firmách využívá MS Excel a právě ten obsahuje jako svou součást VBA. Uživatelé se s programem také rychleji seznámí, neboť s ním budou pracovat přímo v prostředí MS Excel.

Nyní již k popisu praktického využití programu. V této části si uvedeme několik možných situací, které program dokáže vyřešit.

1. Firemní zakázky – Pokud dostaneme ve firmě zakázku, která se bude dotýkat více úseků, potom nám prakticky pomůže vytvořený program.. Je důležité, aby nebylo nezbytně nutné, pevně dané pořadí průchodu zakázky jednotlivými úseky firmy. Prakticky to znamená, že bychom pro každý projekt vytvořili graf, kde vrcholy budou úseky a ohodnocené hrany budou obsahovat časové údaje. Musíme si uvědomit, že z počátečního úseku bychom byli schopni dostat se do více dalších úseků, proto zvolíme ten kde budeme mít nejnižší časovou náročnost, stejným způsobem budeme postupovat z úseků dalších.. Poté bychom nalezli pomocí programu minimální kostru grafu a byli bychom schopni stanovit, jak nejlépe bude daná zakázka procházet jednotlivými úseky, aby byla co nejrychleji dokončena. Jako příklad si můžeme vzít výpočet nákladů na projekt pro externí firmu. Materiály, které budou jednotlivá oddělení využívat se budou postupně rozšiřovat. Výhodou je ovšem skutečnost, že můžeme materiály kopírovat, opisovat a tak dále, proto máme jistotu že v programu budou firemní procesy fungovat.

2. Zavedení kanalizace – Vezměme si případ, kdy dostaneme plán kanalizačního systému, a naším úkolem bude vytvoření vhodné kanalizace. Budeme samozřejmě chtít, aby náklady na výstavbu byly co nejnižší, a zároveň, aby byl odvod co nejrychlejší. K tomu nám opět poslouží náš program, do kterého zadáme jednotlivé vrcholy a ohodnocené hrany mezi danými vrcholy, které představují vyčíslené náklady. Do vyčíslených nákladů musíme zahrnout nejen

vzdálenost mezi danými vrcholy, ale také například složitost terénu. Poté nám již program proběhne a vytvoří se nám minimální kostra grafu, která nám zaručí minimalizaci nákladů a také nejkratší kanalizační odvod.

3. Propojení počítačů – Další oblast, kde můžeme prakticky využít náš program, bude při otevření nové firmy a propojení počítačů. Budeme chtít, aby bylo propojení našich firemních počítačů co nejefektivnější. Do tabulky v Excelu na stanovém listu si vložíme údaje uvádějící vzdálenosti mezi vždy právě dvěma počítači. Poté budeme schopni zjistit jakým způsobem počítače propojit, aby byly náklady na zapojení co nejnižší.

4. Přívod plynu – Představme si situaci, kdy nakoupíme pozemky za účelem vystavení rodinných domů. Zjistíme, že zde není přívod plynu. K této situaci využijeme program a pomocí grafu nalezneme minimální kostru grafu, která nám zajistí ideální zavedení přívodu plynu.

Variant využití programu je mnoho, proto jsem uvedl pouze výčet několika možných příkladů. Další výhodou programu je, že nepotřebujeme vidět zadaný případ graficky, ale stačí nám pouze údaje získané například z tabulky. Pokud ovšem nastane situace, kdy nám program řekne, že graf je nesouvislý, budeme potřebovat i grafické znázornění, abychom mohli zkontrolovat, zda jsou všechny vrcholy a hrany na svém místě a řádně zaznamenány do programu.

Závěr

Grafy a grafové algoritmy nás obklopují ve větší míře, než si dokážeme představit. Cílem této práce bylo představit základní grafové algoritmy a seznámit čtenáře s jejich fungováním a některými příklady využití. Nyní již záleží na samotném čtenáři, jak se získanými poznatky naloží, neboť na některé příklady lze implementovat více algoritmů, proto si můžeme zvolit z více variant, případně je kombinovat a podobně. Základní rozhodnutí, které nesmíme před vytvářením samotného algoritmu opomenout, je seznámení se s daným grafem a daným problémem, který chceme řešit. Proto je nutné orientovat se také v základních pojmech o grafech, které jsou uvedené v této práci. Teprve až se seznámíme blíže s danou situací, můžeme přejít k volbě vhodného algoritmu a zjišťovat pomocí něj konkrétní údaje.

V současné době existuje celá řada algoritmů řešící různé situace od hledání nejkratší cesty v grafu až po hledání minimální kostry grafu. Musíme si ovšem uvědomit, že stejně jako ve světě nových technologií a systémů dochází k novým pokrokům, stejně tak bude docházet k pokrokům i na půdě grafových algoritmů. Je nutné, abychom vnímali algoritmy jako nedílnou součást vývoje již zmíněných technologií a systémů, které nás posunou o krok dále do budoucnosti.

Použitá literatura

1. Knižní zdroje

[1] DEMEL, J. *Grafy a jejich aplikace*. 1.vydání Praha: Academia, 2002. 258 s. ISBN 80-200-0990-6.

[2] MEZNÍK, I. *Diskrétní matematika pro užitou informatiku*. 1.vydání Brno: Akademické nakladatelství CERM, s.r.o., 2009. 104 s. ISBN 978-80-214-3948-1.

[3] NEŠETŘIL, J., MATOUŠEK, J. *Kapitoly z diskrétní matematiky.* Dotisk druhého, opravného vydání Praha: Korolinum, 2003. 381 s. ISBN 80-246-0084-6.

[4] ROSEN, K. H. *Discrete Mathematics and Its Applications*. 4th edition New York : Mc Graw-Hill, 1999. 832 s. ISBN 0-07-289905-0.

2. Elektronické zdroje

[1] ČERNÝ, J. *Základní grafové algoritmy* [online]. Praha : Karlova Univerzita, Matematicko-fyzikální fakulta, 7.9.2008 [cit. 2010-05-11]. Dostupné z WWW: <http://kam.mff.cuni.cz/~kuba/ka/>.

[2] *Eulerovský tah* [online]. 2002 , 25.10.2009 [cit. 2009-11-12]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Eulerovsk%C3%BD_tah>.

[3] JIROVSKÝ, L. *Teorie grafů* [online]. 2008 [cit. 2009-11-12]. Dostupný z WWW: <http://teorie-grafu.elfineer.cz/>.

Seznam obrázků

Přílohy

[1] 1x CD Jarníkův algoritmus